An extension of the arc orienteering problem and its application to cycle trip planning

C. Verbeeck[a,*], P. Vansteenwegen[b], E.-H. Aghezzaf[a]

[a] *Ghent University, Department of Industrial management*
[b] *KU Leuven, Centre for Industrial Management, Traffic and Infrastructure*

Abstract

The cycle trip planning problem (CTPP) can be formulated as a variant of the arc orienteering problem (AOP). The CTPP aims at finding a route with the highest possible score, in a directed graph, among those having a total length that does not exceed some given upper bound. The contributions of this paper are a new mathematical programming model for the CTPP and two solution methods. The first is a branch-and-cut approach that is able to solve small problem instances to optimality and the second is a metaheuristic that solves CTPP and AOP instances of realistic size to near optimality.

Keywords: Arc orienteering problem, Iterated local search, Cycling

1. Introduction

Cycling is becoming increasingly popular as a recreational activity in countries like Belgium, France and the Netherlands. Doing sport is combined with the pleasure of visiting a certain region or enjoying nature. For many people it is a relaxing way to explore and visit their own country or province to discover the surroundings of the place they are visiting. For the local economy cyclists form an important source of income as they spend money for food, beverages, attractions and accommodations during or after the trip [26].

In many regions, large cycling networks are made available by the regional tourist offices. These networks are graphs consisting of vertices (intersections) that are connected by cycle friendly arcs (tracks). From each vertex two or more other vertices can be reached. On all arcs, signposts are placed to guide cyclists from one vertex to the next. The province of East Flanders (Belgium) for instance, maintains a network of five regions that is composed of 989 vertices and 2961 arcs, with a total of 3585 kilometres of tracks. A lot of extra information about the arcs of the network is available: steep inclines or not, dirt or paved roads, car-free or not, etc. As a result, a personal score can be associated with each arc to express the attractiveness of the arc for a given cyclist. Obviously, the attractiveness will be different for different cyclists, since some cyclists might prefer dirt roads with steep inclines and others might prefer paved and flat roads.

Apart from maps which allow for a cumbersome planning processes, one of the products offered by tourist offices to promote their region to cyclists is an online route planning tool. This tool significantly facilitates the complicated planning process of designing an appropriate trip. Cyclists want to design the most attractive trip on the network, tailored to their own personal preferences concerning the length of the trip, the type of arcs, the difficulty, etc. The most advanced planning tool currently available is described by Souffriau et al. [22]. Tourists select the starting point and the preferred length of their trip. Immediately, an appropriate trip is calculated and the cyclist can print this trip or download it to his personal navigation device. However, personal preferences for certain types of arcs or a personal score per arc are not taken into account yet in this tool. Second, apart from other minor limitations, having no lower

*Corresponding author: Ghent University, Technologiepark 903, 9052 Zwijnaarde, Belgium, Tel.: +32 9 264 54 95, Fax.:+32 9 264 58 47
Email addresses: cedric.verbeeck@ugent.be (C. Verbeeck), pieter.vansteenwegen@cib.kuleuven.be (P. Vansteenwegen), elhoussaine.aghezzaf@ugent.be (E.-H. Aghezzaf)

limit on the distance to cycle and forcing the tourists to select the starting point themselves encumbers the planning process and lowers the practical value.

In this paper, the planning functionalities are extended and the underlying optimisation problem becomes more complicated. The preferred length is no longer considered as the objective of the planning problem, but it is replaced by an upper and a lower bound on the total length. The new objective is to maximise the total collected score, related to the personal attractiveness of the travelled arcs. Furthermore, it is allowed to visit the same vertex multiple times, which is not the case in Souffriau et al. [22]'s tool. However, it is not allowed to visit the same arc or its compliment in the opposite direction or the starting vertex more than once. This is considered unpleasant for the cyclist. Another extension considers the selection of the starting point of the cycle trip. In order to avoid cycling again and again on almost the same arcs close to home, very often, cyclists drive by car to one of the parking lots near a more distant cycling network, and start cycling from there. In the tool of Souffriau et al. [22], they had to indicate the starting point of their cycling trip. In this paper, they can select a set of possible starting points and the algorithm automatically selects the best one for their trip. This selection can be based on a perimeter around their home address (maximum 30 minutes driving by car) or it can be all parking lots in the region they want to visit. The result will be a tour of the requested length, with a parking lot in the requested range, that maximizes the pleasure experienced by the cyclist. We call this problem the cycle trip planning problem (CTPP).

Formally, the cycle trip planning problem (CTPP) can be formulated based on an incomplete directed graph, which consists of a set of vertices and a set of arcs. Each arc has a cost (distance or time) and a profit. If two arcs are available in two directions between a pair of vertices, these arcs are called complementary arcs. This corresponds to a road between two vertices that can be travelled in both directions. It should be noted that formulating this problem based on arcs instead of edges allows to assign different profits and even different costs (for instance travel time in case of hills) to an arc and its complementary arc. The objective is to determine a closed tour that maximises the total score and starts and ends in the same starting vertex. The starting vertex should be selected from a set of possible starting vertices. Every possible starting vertex is located on the same location as another regular vertex. It is allowed to visit the same vertex multiple times, but the starting vertex of the tour may only be visited once. In our envisioned application, it would be confusing and unpleasant for the cyclist to pass by his starting point during his trip. For the same reason, each arc (or its complementary arc) may only be travelled once. The total length of the tour has to be in the required range.

Due to the acceptance of smartphones and PDA's with GPS and internet connection, the construction and update of routes, based on new information, increasingly becomes a necessity for a number of tourist applications. More importantly, users expect that this construction and update is done in a very short time span (only a few seconds) and therefore very fast algorithms are required to calculate new routes or update previously scheduled routes.

This paper makes four important contributions to the literature. First, a new and more complex variant of the arc orienteering problem is introduced and mathematically modelled. Special attention goes to the fact that this routing problem has no fixed starting point which increases its practical value for tourist applications. Furthermore, it is allowed to visit vertices multiple times but arcs only once. Together with the non-fixed starting point, this significantly complicates the mathematical formulation and the solution procedures. Secondly, an effective and robust metaheuristic is developed which deals successfully with this problem and also performs well on a set of real-word test problems created and made available for other researchers. This algorithm makes use efficiently of the multi start principle and incorporates various time-savings procedures to speed up the insertion of new arcs to a solution. These principles, as well as the implementation of the insertion itself, can be used as building blocks for future solution methods for any arc orienteering problem. Thirdly, our solution method also outperforms the state-of-the-art solution method developed for the basic arc orienteering problem on a set of benchmark instances. Finally, a branch-and-cut approach is developed that can solve smaller instances of this complex problem to optimality. This also allowed us to verify the performance of our metaheuristic.

In the next section, the available literature on related problems and possible applications is discussed. Section 3 presents the mathematical formulation of the problem. In Section 4 a branch-and-cut approach and a fast and effective solution algorithm for the CTPP are explained. In Section 5, the performance of the algorithms is evaluated based on comprehensive experimental results. In the last section, conclusions and further work are discussed.

2. Literature review

This planning problem is a variant of the Orienteering Problem (OP), a combinatorial optimisation problem in which the total collected score has to be maximised by visiting a selection of locations while not violating the time budget constraint [26, 27]. This model has already been successfully applied in the field of tourism in order to determine personalised walking routes in historic cities [21, 24, 29]. A recent survey about the OP, its practical applications and solution strategies can be found in Vansteenwegen et al. [27].

In arc routing problems in general, the customers are located on arcs and the routes with a minimum cost need to be found. For an overview on arc routing problems we refer to Dror [12]. However, in order to be able to plan cycle routes, an arc routing variant of the OP should be considered. Only a few papers consider arc routing variants of the OP where a profit is associated with each arc instead of each vertex [1–6, 9, 13, 17, 22, 30]. Deitch & Ladany [9] defined an orienteering problem where the objective is to design the route for a touristic bus that maximizes the "attractiveness" of the sites visited and the scenic routes traversed. Feillet et al. [13] proposed a branch-and-price approach for the "profitable arc tour problem". The main difference with the CTPP is that in the profitable arc tour problem every arc can be visited more than once. The application they consider is a tactical freight transportation planning in the car industry where freight trips need to be planned between plants. They also mention reliability in telecommunication network optimisation as an alternative application. Aráoz et al. [3] introduce the "privatised rural postman problem", which was later called the "prize-collecting rural postman problem" by Aráoz et al. [1, 2]. In these problems, the objective is to maximise the difference between the profit and the total travel distance. The most important difference with the CTPP is that these postman problems are defined on an undirected graph and have no constraint on the cost, but include the cost in the objective function. They present the application of collecting recycling bins. When this collection is organised by the public administration, all arcs in an area must be serviced. A private company, however, would only select to serve the streets with the highest profits. One of the latest advances for this particular problem is given by Archetti et al. [5] who proposes an ILP-refined tabu search algorithm that combines a tabu search scheme with Integer Linear Programming.

Ghiani et al. [17] and Archetti et al. [4] describe the "undirected capacitated arc routing problem with profits". The objective of this problem is to determine a route for each available vehicle in order to maximise the total collected profit, without violating the capacity and time limit of each vehicle. The difference between the CTPP and the undirected capacitated arc routing problem with profits is that the (a) CTPP uses a directed graph, (b) considers only one route (vehicle), (c) has no capacity constraints and (d) considers multiple starting points. They consider an application where carriers can select potential customers for transporting their goods. The same application is mentioned by Zachariadis & Kiranoudis [30], but they modify the definition of the undirected capacitated arc routing problem with profits by using a hierarchical objective function that first maximises profit and then explicitly also minimises costs. This is comparable to the profitable arc tour problem and the rural postman problems, but it is different compared to the CTPP.

Souffriau et al. [22] formulate an arc orienteering problem where the goal is to find a tour with a total length as close as possible to a given length. They deal with a simplified version of the cycle trip planning problem. The most important differences with the CTPP are that in the CTPP a vertex can be visited more than once and that in the CTPP a score is associated with each arc, independent of its length.

Archetti et al. [6] describe the team orienteering arc routing problem where a limited number of vehicles needs to service a set of regular customers, while another set of potential customers is available to be serviced. Each customer is associated with an arc of a directed graph and has a certain profit. A fleet of vehicles with a given maximum travelling time is available and routes start and end at a fixed starting point. The main difference is the necessity to visit a set of arcs and the use of multiple vehicles.

Furthermore, other obvious differences between the CTPP and all related problems in the literature are the fact that the starting vertex is not fixed and that a lower bound is imposed on the length of the tour.

The CTPP can not only be used for cycle trip planning, but (parts of) it can also be useful for many of the above mentioned applications. These applications can, for instance, be considered with a set of possible starting points instead of one fixed starting point. In this way, the decision concerning which existing depots to use for particular trips can be optimised together with the routing of the vehicle.

3. Mathematical formulation

The CTPP can be formulated as an integer program. Therefore the cycle network is modelled as an incomplete directed graph $G = (V, A)$, where V represents the set of vertices (v) and A the set of arcs (a). Next, each arc has a cost c_a and a profit p_a and a complementary arc $\overline{a}$. In this formulation, $\delta(S)$ represents the set of outgoing arcs from set S to $V \setminus S$ and $\lambda(S)$ represents the set of incoming arcs in set S from $V \setminus S$. The starting vertex d of the tour should be selected from a set of possible starting vertices called $D \subseteq V$. The minimum and maximum tour length is displayed as C_{min} and C_{max}.

The decision variables are x_a (=1 if arc a is travelled; 0 otherwise), z_v (the number of times vertex v is visited) and y_d (=1 if start vertex d is selected as the start vertex of the route; 0 otherwise).

$$\text{Max} \sum_{a \in A} p_a * x_a \tag{1a}$$

$$\sum_{a \in A} c_a * x_a \leq C_{max} \tag{1b}$$

$$\sum_{a \in A} c_a * x_a \geq C_{min} \tag{1c}$$

$$\sum_{a \in \lambda(v)} x_a - \sum_{a \in \delta(v)} x_a = 0 \qquad \forall v \in V \tag{1d}$$

$$\sum_{a \in \delta(v)} x_a = z_v \qquad \forall v \in V \tag{1e}$$

$$\sum_{d \in D} y_d = 1 \tag{1f}$$

$$\sum_{a \in \delta(S)} x_a + (1 - y_d) \geq \frac{\sum_{v \in S} z_v}{\sum_{v \in S} |\delta(v)|} \qquad \forall d \in D, S \subseteq V \setminus \{d\} \tag{1g}$$

$$\sum_{a \in \delta(d)} x_a \leq (|\delta(d)| - 1) * (1 - y_d) + 1 \qquad \forall d \in D \tag{1h}$$

$$y_d \leq \sum_{a \in \delta(d)} x_a \qquad \forall d \in D \tag{1i}$$

$$x_a + x_{\overline{a}} \leq 1 \qquad \forall a \in A : \exists \overline{a} \in A \tag{1j}$$

$$x_a \in \{0, 1\} \tag{1k}$$

The objective function (1a) maximises the total collected score. Constraints (1b) and (1c) enforce the length of the tour to be within the range $[C_{min}, C_{max}]$. Constraints (1d) and (1e) state that if a vertex is part of the solution, the number of incoming arcs has to be equal to the number of outgoing arcs and equal to the number of times the vertex is visited. Constraint (1f) guarantees that exactly one starting vertex is selected.

The sub-tour Constraints (1g) in the model specify that if collection S contains a node that is part of the solution at least once, there has to be at least one outgoing arc from S to $V \setminus S$ (reduced to $\delta(S)$ in the problem formulation). As a consequence of the even degree constraints (1d), there will also be an outgoing arc for every incoming arc in a set S. These sub-tour elimination constraints are based on the formulation of Dantzig et al. [7]. It is well known that the number of these constraints increases exponentially with an increasing number of vertices. However, the alternative sub-tour elimination constraints introduced by Miller et al. [20] for the travelling salesperson problem, cannot be used for this problem. Since vertices can be visited more than once in the problem under study, it is not possible to order the vertices in the tour, as proposed by Miller et al. [20]. Note that the sub-tour elimination constraints are repeated for every starting vertex as each set S does not incorporate one of the starting vertices since each starting vertex can also be incorporated into a route as a normal vertex.

In order to explain Constraints (1h) and (1i), it should be repeated first that all starting vertices (d) are located on the same location as regular vertices (v_d). This implies that if (and only if) vertex d is selected as starting vertex, vertex

v_d may not be visited. If vertex d is not selected as starting vertex, v_d can be visited multiple times. This condition is imposed by constraints (1h). Constraints (1j) guarantee that if arcs are available in two directions between two vertices, at most one of these two arcs is travelled. In this way, travelling back and forth on the same road is avoided. In practice, for cyclists, almost always arcs will be available in two directions.

In Figure 1 an example of a CTPP instance is displayed. In the upper part of the figure the graph consisting of 10 vertices and 15 arcs is displayed. For simplicity reasons the arcs are assumed to have symmetrical scores and distances. In the lower part the optimal solution with objective function equal to 25 is displayed for this problem. Starting vertex 7 is selected as starting point of the tour and the route passes through the other starting vertex 5.

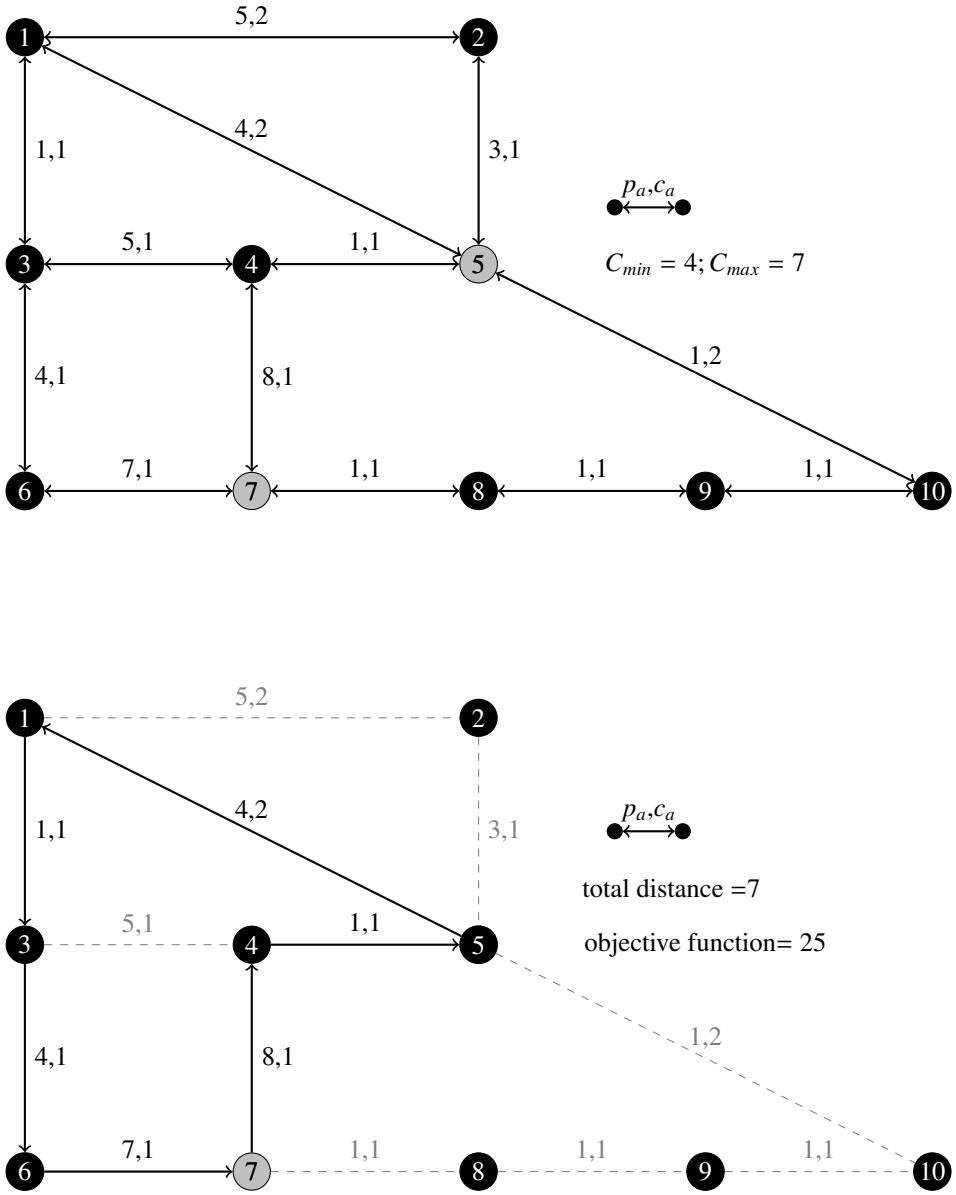

Figure 1: Example CTPP instance with optimal solution

4. Solution approach

The CTPP is a difficult problem to solve, and since the envisioned application requires a solution in only a few seconds, exact solution techniques may appear less competitive for large networks. The mixed integer problem formulation presented in Section 3 can only be solved for very small instances like the example displayed in the corresponding section because the amount of sub-tour constraints increases exponentially with the number of vertices. However, its solutions will allow us to assess any heuristic proposed for this problem.

In Section 4.1 a branch-and-cut approach is designed to tackle the exponentially increasing amount of sub-tour constraints. As will be shown in Section 5 the branch-and-cut approach is not able to solve large instances within an acceptable computational time. Therefore a metaheuristic solution approach was developed and discussed in Section 4.2.

4.1. Branch-and-cut algorithm

Branch-and-cut is a well-known technique already used to obtain optimal solutions for small sized orienteering problems, for example by Figliozzi [14], Gendreau et al. [16] and Feillet et al. [13]. Therefore a branch-and-cut algorithm was implemented and tested for the CTPP.

Branch-and-cut for integer linear programming combines the usual branch-and-bound method with cutting plane procedures. At each node of the branch-and-bound tree the LP-solution set is strengthened using some valid inequalities to improve the lower bounds. In our case, the problem formulation of the CTPP proposed in Section 3 has an exponential number of sub-tour elimination constraints (Constraints 1g). Initially, none of the sub-tour elimination constraints are used to solve the LP relaxation of the problem. Instead, a separation procedure is developed and used to generate violated sub-tour elimination constraints, which are then added to the problem formulation. We applied the same separation strategy for the sub-tour elimination constraints as proposed by Dantzig et al. [7] and Dantzig et al. [8]. After an LP solution has been found, a dummy graph $G^\star = (V^\star, A^\star)$ is created based on all the vertices and arcs that correspond to the non-zero z and x variables. The actual x values are used as capacities on the arcs of the dummy graph. The start vertex with the highest fractional value for its y variable is chosen as the starting vertex $d^\star$. The maximum flow/minimum cut problem is solved on $G^\star$ using the algorithm of Hao & Orlin [18]. If a minimum cut smaller than 1 is found, the LP solution can be cut off by one or more sub-tour elimination constraints. Therefore the sub-tour elimination constraints (1g) are added to the current problem formulation with $d^\star$ as starting vertex and $S^\star$ equal to all the vertices that are cut off, directly or indirectly, from $d^\star$. Afterwards the solver tries to find a new LP solution that satisfies the newly added constraints and the above process is repeated again. Once no violated sub-tour constraints can be generated, the problem is solved as an integer problem using the usual branch-and-bound method. The separation procedure is again used on subsequently obtained integer solutions to verify sub-tour correctness and adds additional constraints if necessary. The cutting plane procedure is displayed in Algorithm 1.

Algorithm 1 Cutting plane procedure(LP solution: x,z,y)

Create empty dummy graph $G^\star = (V^\star, A^\star, \textit{starting point } d^\star)$
for v $\in V$ **do**
  if $z_v > 0$ **then**
    add v to $G^\star$
  end if
end for
for a $\in A$ **do**
  if $x_a > 0$ **then**
    add a to $G^\star$
    capacity of $a = x_a$
  end if
end for
for d $\in D$ **do**
  $d^\star = max(y_d)$
end for
mincut $\leftarrow$ min cut algorithm on $G^\star$
if mincut ≥ 1 **then**
  stop procedure
end if
$S^\star \leftarrow$ vertices cut from $d^\star$
if $\frac{\sum_{v \in S^\star} z_v}{\sum_{v \in S^\star} |\delta(v)|} + y_{d^\star} - 1 > \sum_{a \in \delta(S^\star)} x_a$ **then**
  add $\sum_{a \in \delta(S^\star)} x_a + (1 - y_{d^\star}) \geq \frac{\sum_{v \in S^\star} z_v}{\sum_{v \in S^\star} |\delta(v)|}$ to LP formulation
end if

4.2. Metaheuristic solution method

Since high-quality solutions are required and the computation time should be limited to only a few seconds, the literature on vehicle routing suggests the implementation of a local search based metaheuristic [25]. In this paper, an iterated local search framework is implemented to tackle the CTPP. ILS is described by Lourenço et al. [19]. A sequence of local search solutions is built up, instead of repeating random local search trials. A good balance between intensification of the search and diversification is essential to be successful. The choice for this framework was motivated by the fact that generally very complex problems require simple solution frameworks and ILS has proven to be successful for other variants of the node orienteering problem [15, 23, 28] but was not used on the arc-variant yet.

The proposed solution approach consists of an initialisation phase to obtain a set of feasible solutions and an improvement phase implementing the iterated local search framework for each solution in this set. In the specific implementation of the solution algorithm, the first phase is applied for each possible starting point. Thereafter, the four best solutions from any of these starting points are retained. The improvement phase is then executed on each of these four solutions and the best result is used as the final result. During this improvement phase, a part of the solution is removed in every iteration and the insert move is used to close the gap and to increase the total score.

Algorithm 2 Outline of the metaheuristic solution method

for all starting locations **do**
  Initialisation (Algorithm 4)
end for
retain four best initial solutions
for all retained initial solutions **do**
  Improve (Algorithm 5)
end for
select best solution

Before discussing both phases in detail, the ”insert move”, applied in both phases, is explained.

4.2.1. Insert move

The goal of the insert move is to close a ”gap” in the tour. This means a starting part of a solution and an ending part are available, but these parts are not connected yet. Therefore, a gap can be defined by a starting vertex and an ending vertex. Closing this gap is done by inserting a set of arcs which increases as much as possible the total collected score, without violating the allowed range on the total length. It is important to notice that the insert move is not looking for the shortest path between the starting and ending vertex, but aims at the highest scoring path. This is much more difficult than the shortest path problem and it is actually a smaller version of the CTPP problem itself.

The insert move is implemented as a recursive procedure. After inserting one of the possible arcs leaving from the current vertex (`start`), the insert move is applied again, but now it starts from the end of the arc that was just inserted. Obviously, the available distance (`dist`) to close the gap is reduced by the length of the inserted arc. The search structure corresponds to a “depth first” search in a search tree. If more than one arc can be inserted after a given vertex, one is selected to be inserted. The others will only be considered after the whole branch of this arc was examined. This procedure discussed until now, would be a very time (and memory) consuming process which is unacceptable for the application under consideration. Therefore, a few time saving procedures are designed.

The first time saving procedure reduces the number of arcs that are considered for insertion. After insertion, it should always be possible to close the remaining gap without violating the constraint on the total length (C_{max}). Therefore, only arcs are considered (`availableArcs`) of which the length of the shortest path between the end vertex of the arc and the end vertex of the gap is smaller than the remaining available distance. The shortest path between each pair of vertices is calculated once “off-line”, before the planning tool is actually used, and stored in a database. Therefore, calculating the shortest paths does not slow down the calculation of the cycle trip. On the contrary, the length of any shortest path, and the arcs comprised in the shortest path, will speed up the solution algorithm.

The second time saving procedure uses a parameter (`maxDepth`) that restricts the depth of the search. This parameter indicates the maximum number of arcs that can be considered to close a gap. The third procedure is called “first improvement local search”. The insert move ends as soon as a feasible solution is obtained in any of the branches of the search tree.

Next to the distance that is still available (`dist`) and the maximum number of arcs to consider (`maxDepth`) an extra parameter is added to the insert move, but only when it is used to improve an existing solution. The sum of the scores of the removed arcs (`minProfit`) is used as a lower bound on the score that should be collected during the insert move. In other words, in the improvement phase, the insert move will not stop before a solution is found that increases the total collected score and respects both upper (C_{max}) and lower bound (C_{min}) on the total length.

Finally, it should be noted that in most cases in a real-life road network, only four roads come together at a vertex. In a cycle network, very often only three cycle friendly arcs come together, of which one is complementary to the one leading to the vertex and should therefore not be considered. As a result, at most vertices, only two arcs have to be examined. This significantly reduces the maximum size of the search tree: from around $N^{maxDepth}$ in a theoretical complete graph (with N vertices) to around $2^{maxDepth}$ in a real-life cycle network.

The insert move as well as the incorporated time-savings procedures to speed up the insertion of new arcs to a solution can be used as building blocks for future solution methods for any arc orienteering problem.

An outline of the complete insert move is presented in Algorithm 3. The insert move is implemented as a recursive boolean function, therefore the actual solution can be found in tempSolution if the result of the function is true.

Algorithm 3 Insert(tempSolution,`start`,`end`,`dist`,`minProfit`,`maxDepth`)

```
if (maxDepth < 0)||shortestPath(start,end) > dist then
  return false
else
  for each arc ∈ availableArcs do
    add arc to tempSolution
    if (end of arc = end)&&(length of tempSolution ≥ C_min)&&(score of tempSolution> minProfit) then
      return true
    else
      if Insert(tempSolution,end of arc,end,dist-length of arc,minProfit,maxDepth-1) then
        return true
      end if
    end if
    remove arc from tempSolution
  end for
end if
return false
```

4.2.2. Initialisation phase

The goal of the initialisation phase is to design a first feasible solution. The initialisation phase has a straightforward implementation based on the insert move. The insert move is applied between the end of each possible "starting arc" leaving from the starting vertex, and the starting vertex itself. No limit is imposed on the number of arcs that can be used to close this gap, however, the insert move will end as soon as a feasible solution for the given starting arc is obtained. Obviously, a feasible solution should respect both upper (C_{max}) and lower bound (C_{min}) on the total length. Then, one by one, each arc is removed and the insert move tries to replace this arc and increase the total score of the solution at the same time.

Finally, the solutions for the different starting arcs are compared and the four best solutions are retained as starting solutions for the improvement phase. The initialisation phase is summarised in Algorithm 4. Preliminary experiments pointed out that there was no benefit in retaining more than four solutions for the improvement phase.

A first feasible solution generated by the initialisation phase to the problem discussed in Section 3 is displayed in Figure 2. The solution sequence equals $\{[7;4],[4;3],[3;6],[6;7]\}$. Subsequently, if during the removal of one arc and reinsertion part of this algorithm arc $[4;3]$ is removed the subsequent insert move will find the optimal solution with sequence $\{[7;4],[4;5],[5;1],[1;3],[3;6],[6;7]\}$ which is displayed in Figure 1.

Algorithm 4 Initialisation(`maxDepth`,startlocation)

```
Solution ← empty solution
if Insert(Solution,startlocation,startlocation,C_max,0,maxDepth) then
  for each arc_i ∈ tempSolution do
    tempSolution ← empty solution
    tempSolution ← Solution
    remove arc_i from tempSolution
    minProfit=score of arc_i
    if Insert(tempSolution,end of arc_{i-1},start of arc_{i+1},C_max−length of tempSolution,minProfit,maxDepth) then
      Solution←tempSolution
    end if
  end for
end if
return Solution
```

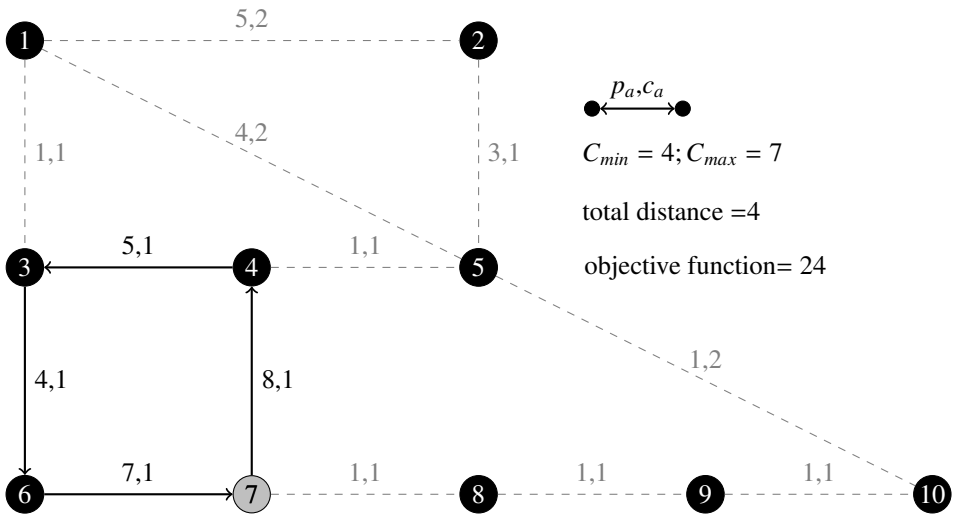

Figure 2: First feasible solution

4.2.3. Improvement phase

The improvement phase is a specific implementation of the Iterated Local Search (ILS) framework. ILS is described by Lourenço et al. [19]: a sequence of local search solutions is built up, instead of repeating random local search trials. A good balance between intensification of the search and diversification is essential to be successful. ILS has proven to work well on the node-variant of the orienteering problem [23, 28] but was not used on the arc-variant yet. In the current implementation, we will remove, in every iteration of the improvement phase, a part of the solution and use the insert move to close the gap and increase the total score.

The removal of the arcs is based on two variables, the first one indicates the number of consecutive arcs to remove from the solution (`R`) and the second one indicates the position in the solution sequence (`A`) to start the removal of arcs. Both variables start with a value of one, and every iteration, the value of `A` and `R` is increased by one. If the removal reaches the end vertex of the solution, which is equal to the starting vertex, the removal is stopped and `A` is reset to one for the next improvement iteration. Furthermore, if at the start of an improvement iteration `R` is bigger than the amount of arcs in the solution, `R` is reset to 1.

By using variables `R` and `A` in this way, other arcs are removed during every iteration and during the whole improvement phase, most likely, every arc is removed at least once. A similar removal technique was successful in previous implementations of ILS for the node-variant of the orienteering problem [23, 28].

The insert move is applied to close the gap caused by the removal. `R` and `A` are reset to one when a new best solution is found. The iterations of the improvement phase are stopped when no improvement of the current solution is found during `maxNoImprove` consecutive iterations. The improvement phase, displayed in Algorithm 5, is performed for each of the four starting solutions retained from the initialisation phase. Out of the four improved solutions the one with the best score is selected and returned by the algorithm as the final solution to the CTPP problem.

For the solution displayed in Figure 2 the improvement iteration with $A = 2$ and $R = 1$ would lead to the same optimal solution as displayed in Figure 1.

Algorithm 5 Improve(Solution,`maxNoImprove`,`maxDepth`)

```
A=1,R=1,noimprove=0
while noimprove < maxNoImprove do
  tempSolution ← empty Solution
  tempSolution ← Solution
  size= ← number of arcs in tempSolution
  if R > size then
    R = 1
  end if
  start = A
  if A + R > size − 1 then
    end=starting vertex of tempSolution
    remove all arcs starting from A until the end of tempSolution
    A = 1
  else
    end = A + R
    remove R arcs from tempSolution starting from location A
  end if
  minProfit=sum of the scores of the removed arcs
  if Insert(tempSolution,start,end,C_max-length of tempSolution,minProfit,maxDepth) then
    Solution=tempSolution
    noimprove=0
    A=1,R=1
  else
    noimprove=noimprove+1
    A=A+1
    R=R+1
  end if
end while
```

5. Computational experiments

5.1. Instance creation and experimental setup

Three sets of test instances are used for the computational experiments. All these sets are based on the real-life cycle network of East-Flanders ([22]) that consists of 989 vertices and 2961 arcs, with a total of 3585 kilometres of tracks.

So far no benchmark instances have been developed for the CTPP. Therefore we have created 3 new datasets based on 50 AOP instances created by Souffriau et al. [22]. In these instances the score of each arc was set equal to the distance of the particular arc. Furthermore, in the AOP problem formulation the starting vertex for each instance is fixed and there is no minimum distance required in order to obtain a feasible route. In the AOP it is not allowed to visit a vertex twice. In our CTPP formulation all vertices can be visited more than once, but the starting vertex can only be visited at the start and the end of the trip.

The 3 new CTPP datasets and the corresponding results are discussed in the following sections. Finally, the cycle network and all these instances together with their known optimal solutions can be found at:
https://www.mech.kuleuven.be/en/cib/op/

The Mixed integer problem formulation of the AOP was implementend using the commercial solver CPLEX 12.5 (64-bit) on a computer with an i7-3770 3.4 GHz processor and 32 GB of memory. The branch-and-cut algorithm for the CTPP was implemented in C++ using CPLEX 12.5. For the implementation of the minimum cut algorithm of [18] the C++ library LEMON 1.3 by [10] was used. The tests were performed on a computer with an i5-2540M 2.6 GHz processor and 8 GB of memory. The ILS was programmed in C++ (using OMP for parallel computing) and the

tests were performed on a computer with an i5-2540M 2.6 GHz processor and 8 GB of memory. The shortest paths (Section 4.2.1) were calculated before the start of the ILS, using Dijkstra's algorithm.

5.2. *Input parameter fine-tuning*

The ILS has only two input parameters namely `maxNoImprove` and `maxDepth`. Table 1 shows the effect of ranging values of `maxDepth` on the performance on the instances of Section 5.4. As a performance metric, the CPU-time is used, together with the percentage difference between the total score of the optimal solution and the total score of the metaheuristic solution:

$$\%\ \text{diff} = \frac{\text{score CPLEX} - \text{score ILS}}{\text{score CPLEX}} * 100 \tag{2}$$

The CPU time increases when using values higher than 30 while at the same time the solution quality does not improve significantly. Therefore a value of 30 is selected to perform all the experiments. Note that for values smaller than 20 some instances could not be solved as the starting vertex could not be reached using a path with a length $\geq C_{min}$.

Second, the effect of `maxNoImprove` is shown in Table 2. As more improvement iterations are performed, the computational time increases and the solution quality increases. However, the increase in solution quality decreases when values above 30 are chosen. Therefore a value of 30 is selected for this input parameter.

Table 1: The effect of `maxDepth` while `maxNoImprove`=30

`maxDepth`	Diff (%)	# instances solved	CPU(s)
5	2.345%	10	0.0
10	0.698%	26	0.0
15	0.042%	43	0.0
20	0.001%	49	0.1
30	0.000%	49	0.5
40	0.000%	49	2.4

Table 2: The effect of `maxNoImprove` while `maxDepth`=30

`maxNoImprove`	Diff (%)	CPU(s)
5	2.594%	0.0
10	0.645%	0.0
20	0.009%	0.0
30	0.000%	0.5
40	0.000%	30.1

5.3. *35 new CTPP instances solved by branch-and-cut and ILS*

For the first experiment, we created 35 new CTPP instances by randomly selecting 8 starting vertices out of the 10 available starting vertices used in the AOP instances of Souffriau et al. [22]. Furthermore, instead of using the length of an arc as its score, we assigned a random score between 0 and 10 to every arc, while ensuring that complementary arcs have the same score. These CTPP instances were solved by the branch-and-cut algorithm and the ILS.

The results of the branch-and-cut algorithm and the ILS are compared in Table 3 where the first column lists the instance name, the second the possible starting ids and the third column the upper limit on the travel time. The fourth column states the optimal score found by the branch-and-cut method together with the corresponding starting id and computation time in the two succeeding columns. The last three columns contain the ILS solution consisting of the score, computation time and % difference with the optimal branch-and-cut solution. Note that not all instances could be solved by the branch-and-cut algorithm due to time limitations. We conclude that the ILS was able to find all optimal solutions which could be found by the branch-and-cut algorithm with an average computation time of 0.10 seconds which is obviously much faster than the average computation time of 4541 seconds of the branch-and-cut approach. The results of the ILS for the instances that could not be solved by branch-and-cut are also displayed.

The computational time increases for instances with a maximum distance limit of 140 kilometres. As the search tree becomes too large, the first insert move of the initialisation phase which tries to find a feasible solution starting from the starting location back to the starting location, spends a large proportion of the computation time. If we lower the `maxDepth` value in order to reduce the search tree, we run the risk of not finding a solution since a feasible solution for this problem setting can easily contain more than 25 arcs. However, this does not diminish the practical value of the solution method since almost all routes requested by recreational cyclists will have a maximal distance much lower than 140 kilometres.

Table 3: 35 new CTPP instances solved by branch-and-cut and ILS

			Branch-and-cut			ILS		
name	ids	C_{max}	S	start	CPU(s)	S	CPU(s)	Diff (%)
20.1	26,18,10,30,14,22,6,2	20000	75	10	28	75	0.0	0.0
20.2	38,14,18,30,22,10,2,26	20000	75	10	55	75	0.0	0.0
20.3	2,6,26,30,14,34,38,18	20000	63	34	314	63	0.0	0.0
20.4	22,6,26,30,14,18,10,34	20000	75	10	447	75	0.0	0.0
20.5	6,22,34,2,18,30,26,38	20000	63	34	804	63	0.0	0.0
40.1	26,18,10,30,14,22,6,2	40000	188	10	944	188	0.0	0.0
40.2	38,14,18,30,22,10,2,26	40000	188	10	1076	188	0.0	0.0
40.3	2,6,26,30,14,34,38,18	40000	151	34	11860	151	0.0	0.0
40.4	22,6,26,30,14,18,10,34	40000	188	10	12377	188	0.0	0.0
40.5	6,22,34,2,18,30,26,38	40000	151	34	24229	151	0.0	0.0
60.1	26,18,10,30,14,22,6,2	60000	289	10	336	289	0.1	0.0
60.2	38,14,18,30,22,10,2,26	60000	289	10	2029	289	0.5	0.0
avg					4541		0.1	0.0
60.3	2,6,26,30,14,34,38,18	60000	246	6	$> 100h$	246	0.4	0.0
60.4	22,6,26,30,14,18,10,34	60000	289	10	$> 100h$	289	0.1	0.0
60.5	6,22,34,2,18,30,26,38	60000	246	6	$> 100h$	246	0.4	0.0
80.1	26,18,10,30,14,22,6,2	80000	na	na	na	331	0.5	na
80.2	38,14,18,30,22,10,2,26	80000	na	na	na	331	0.5	na
80.3	2,6,26,30,14,34,38,18	80000	na	na	na	324	0.7	na
80.4	22,6,26,30,14,18,10,34	80000	na	na	na	331	0.6	na
80.5	6,22,34,2,18,30,26,38	80000	na	na	na	324	0.2	na
100.1	26,18,10,30,14,22,6,2	100000	na	na	na	393	1.3	na
100.2	38,14,18,30,22,10,2,26	100000	na	na	na	393	1.7	na
100.3	2,6,26,30,14,34,38,18	100000	na	na	na	385	1.6	na
100.4	22,6,26,30,14,18,10,34	100000	na	na	na	393	1.4	na
100.5	6,22,34,2,18,30,26,38	100000	na	na	na	385	0.4	na
120.1	26,18,10,30,14,22,6,2	120000	na	na	na	441	1.521	na
120.2	38,14,18,30,22,10,2,26	120000	na	na	na	441	1.606	na
120.3	2,6,26,30,14,34,38,18	120000	na	na	na	459	1.675	na
120.4	22,6,26,30,14,18,10,34	120000	na	na	na	459	1.722	na
120.5	6,22,34,2,18,30,26,38	120000	na	na	na	459	0.446	na
140.1	26,18,10,30,14,22,6,2	140000	na	na	na	480	11.564	na
140.2	38,14,18,30,22,10,2,26	140000	na	na	na	480	10.528	na
140.3	2,6,26,30,14,34,38,18	140000	na	na	na	480	10.019	na
140.4	22,6,26,30,14,18,10,34	140000	na	na	na	480	10.868	na
140.5	6,22,34,2,18,30,26,38	140000	na	na	na	455	3.897	na
avg							1.8	

5.4. 50 original AOP instances solved by ILS

Since not all CTPP instances could be solved by the branch-and-cut algorithm within the given time frame, additional experiments were conducted to verify the performance of the ILS. The 50 original AOP instances were solved by the ILS and the performance was compared against the GRASP and the commercial solver implementations published by Souffriau et al. [22]. In order to do a fair comparison ILS was ran on the AOP instances with a fixed starting vertex and C_{min} was set equal to $\frac{C_{max}}{2}$. Subsequently, we kept track if a vertex is visited more than once in the final

CTPP solution. Since it was not allowed to visit a vertex more than once in the AOP problem formulation, it is possible to obtain a higher score than in the original AOP for these particular instances (denoted by a star symbol (*) in Table 4 next to the obtained result).

With our implementation of the mixed integer linear problem formulation of the AOP we obtained a number of different optimal solutions than Souffriau et al. [22]. The biggest difference was observed for the two instances with id 6 and 26 and a C_{max} equal to 20,000. We suppose that the reason for this is that a symmetric distance between nodes is assumed in the implementation of Souffriau et al. [22] which is not the case for the real-life cycle network of East-Flanders and our implementation. This slightly different network also explains why the GRASP obtained solutions with a higher score than our optimal solution, on the instances with id 10 and 22 and a C_{max} equal to 20000 and the instance with id 26 and a C_{max} equal to 40000 (indicated with a ! in Table 4) . We were not able to find optimal solutions for the instances with a $C_{max} \geq 60000$ but since our ILS is able to find solution with a cost almost equal to C_{max} we assume the optimal objective function to be equal to C_{max}.

The detailed results are displayed in Table 4. The first column lists the id, the second column the upper bound on the travel time, the third and the fourth the optimal solution and the computation time needed by CPLEX. The score and % difference with the optimal solution of the GRASP and ILS solution methods are displayed in the succeeding columns. The instance with start id 30 and C_{max} equal to 20,000 has no feasible solution and is therefore not taken into account during the analysis of the results. Our algorithm has a gap of 0.0% in comparison with the optimal solution and the GRASP approach of Souffriau et al. [22] has a gap of 0.40% with the optimal solution. Moreover, it should be noted that our ILS obtains an average computation time of 0.6 seconds which is comparable to the GRASP computation time of at most 1 second and obviously much faster than the commercial solver (47386 seconds).

Table 4: 50 original AOP instances solved by ILS

		CPLEX		GRASP		ILS	
Id	C_{max}	S	CPU(s)	S	Diff (%)	S	Diff (%)
2	20000	19497	22	19495	0.0	19497	0.0
6	20000	18778	109	15874	15.5	18778	0.0
10	20000	19711!	128	19712	0.0	19711	0.0
14	20000	19918	178	19918	0.0	19918	0.0
18	20000	19602	190	19602	0.0	19602	0.0
22	20000	19564!	221	19565	0.0	19564	0.0
26	20000	19919	251	19871	0.2	19919	0.0
30	20000						
34	20000	19944	326	19943	0.0	19944	0.0
38	20000	19132	374	19131	0.0	19132	0.0
2	40000	39998	420	39948	0.1	39998	0.0
6	40000	39996	537	39930	0.2	39996	0.0
10	40000	39976	15157	39941	0.1	39976	0.0
14	40000	39994	75505	39970	0.1	39994	0.0
18	40000	39993	94181	39706	0.7	39993	0.0
22	40000	39982	132857	39978	0.0	39982	0.0
26	40000	39996!	132920	39997	0.0	39996	0.0
30	40000	39992	137113	39571	1.1	39992	0.0
34	40000	39999	137210	39932	0.2	39999	0.0
38	40000	39995	172641	39967	0.1	39995	0.0
2	60000	60000	na	59980	0.0	60000	0.0
6	60000	60000	na	59982	0.0	60000	0.0
10	60000	60000	na	59989	0.0	60000	0.0
14	60000	60000	na	59997	0.0	60000	0.0

18	60000	60000	na	59973	0.1	60000	0.0
22	60000	60000	na	59913	0.2	60000	0.0
26	60000	60000	na	59988	0.0	60000	0.0
30	60000	60000	na	59799	0.3	59999	0.0
34	60000	60000	na	59993	0.0	60000	0.0
38	60000	60000	na	59992	0.0	60000	0.0
2	80000	80000	na	79974	0.0	80000	0.0
6	80000	80000	na	79977	0.0	80000	0.0
10	80000	80000	na	79997	0.0	80000	0.0
14	80000	80000	na	79989	0.0	80000	0.0
18	80000	80000	na	79943	0.1	80000	0.0
22	80000	80000	na	79969	0.0	80000*	0.0
26	80000	80000	na	79983	0.0	80000	0.0
30	80000	80000	na	79977	0.0	80000	0.0
34	80000	80000	na	79997	0.0	80000	0.0
38	80000	80000	na	79882	0.2	80000	0.0
2	100000	100000	na	99992	0.0	100000*	0.0
6	100000	100000	na	99989	0.0	100000	0.0
10	100000	100000	na	99952	0.1	100000	0.0
14	100000	100000	na	99965	0.0	100000	0.0
18	100000	100000	na	99998	0.0	100000	0.0
22	100000	100000	na	99997	0.0	100000	0.0
26	100000	100000	na	99917	0.1	100000*	0.0
30	100000	100000	na	99884	0.1	100000	0.0
34	100000	100000	na	99998	0.0	100000	0.0
38	100000	100000	na	99912	0.1	100000	0.0
Overall	avg		47386		0.40%		0.00%

5.5. 50 modified AOP instances solved by ILS

The most important disadvantage of the instances used in Section 5.4 is that the score of each arc corresponds to its length. In this third experiment, we will use our implementation of the MILP formulation of the AOP to solve these instances after assigning the same random score to each arc as in experiment 1. The comparison in performance for the third set of instances is shown in Table 5. The first column lists the id, the second column the upper bound on the travel time, the third and fourth the optimal AOP solution and the computation time needed by CPLEX. Then, for the ILS, the score, the computation time, the % difference with the optimal AOP solution and the number of arcs included in the final solution are displayed in the succeeding columns. Note that not all AOP instances could be solved with the commercial solver due to the complexity and the current basic formulation of the MILP for AOP. The computation time needed to solve some of the instances with $C_{max} = 60000$ exceeded 7 days. Therefore we did not try to solve instances with a higher value of C_{max}. To enable future comparison the results of the ILS for instances with no known optimal AOP solution were also included.

The results prove the high performance quality of the ILS since the average gap is very low, 0.3%. Furthermore, the known optimal solution could be found for 27 out of 30 test instances. 4 instances have a single vertex that is visited twice and are marked with a star in Table 5.

Table 5: 50 modified AOP instances solved by ILS

		CPLEX		ILS			
id	C_{max}	S	CPU(s)	S	CPU(s)	Diff (%)	# arcs
2	20000	**54**	16	54	0.0	0.0%	13

6	20000	**58**	49	58	0.0	0.0%	10
10	20000	**75**	59	75	0.0	0.0%	11
14	20000	**54**	85	54	0.0	0.0%	10
18	20000	**30**	104	30	0.0	0.0%	7
22	20000	**44**	122	44	0.0	0.0%	7
26	20000	**48**	164	48	0.0	0.0%	9
30	20000						
34	20000	**63**	234	63	0.0	0.0%	11
38	20000	**60**	268	60	0.0	0.0%	10
2	40000	**146**	231	146	0.0	0.0%	28
6	40000	**139**	4336	139	0.0	0.0%	23
10	40000	**188**	4375	188	0.0	0.0%	26
14	40000	**151**	4908	151	0.0	0.0%	28
18	40000	**110**	2362	110	0.0	0.0%	16
22	40000	**118**	4021	118	0.0	0.0%	22
26	40000	**139**	4331	139	0.0	0.0%	23
30	40000	**94**	4915	94	0.0	0.0%	17
34	40000	**151**	6072	151	0.0	0.0%	22
38	40000	**138**	7006	138	0.0	0.0%	22
2	60000	**238**	577	238	0.0	0.0%	44
6	60000	**246**	5135	246	0.0	0.0%	38
10	60000	**289**	5221	289	0.0	0.0%	45
14	60000	**230**	11647	229	0.0	0.4%	39
18	60000	**186**	222260	186	0.1	0.0%	28
22	60000	**186**	370767	184*	0.1	1.1%	34
26	60000	**246**	371518	228	0.0	7.3%	36
30	60000	**163**	416174	163	0.0	0.0%	28
34	60000	**230**	446642	230	0.0	0.0%	39
38	60000	**213**	612976	213	0.3	0.0%	31
	Avg		86434		0.0	0.3%	
2	80000	na	na	305	0.0	na	53
6	80000	na	na	307	0.0	na	51
10	80000	na	na	331	0.1	na	52
14	80000	na	na	275	0.3	na	53
18	80000	na	na	238	0.2	na	43
22	80000	na	na	252*	0.1	na	40
26	80000	na	na	308	0.1	na	46
30	80000	na	na	241	0.0	na	41
34	80000	na	na	324*	0.1	na	50
38	80000	na	na	310	0.1	na	53
2	100000	na	na	347	0.1	na	60
6	100000	na	na	368	0.0	na	62
10	100000	na	na	393	0.1	na	64
14	100000	na	na	328	1.2	na	58
18	100000	na	na	312	0.1	na	54
22	100000	na	na	326	0.1	na	54
26	100000	na	na	379*	0.0	na	64

30	100000	na	na	338	0.1	na	53
34	100000	na	na	364	0.1	na	58
38	100000	na	na	385	0.3	na	59

6. Conclusion and further work

Traditional route planners only apply shortest time or distance optimisation objectives to their route planning problems. This paper presents a novel approach to trip planning in a directed graph, namely finding the route with the highest score, departing from one of the possible starting locations and with a length between a minimum and maximum distance. The problem is a combinatorial optimisation problem, called the cycle trip planning problem (CTPP). The CTPP is mathematically modelled and solved using two solution methods for instances that are based on the real-life cycle network of the province of East-Flanders in Belgium. This network together with the test problems is also made available to allow comparison with future research. A branch-and-cut approach is able to solve small problem instances and an ILS metaheuristic solves CTPP instances to near optimality in less than 2 second on average. This performance is due to the multi start principle and various time-savings procedures in the insertion move. The implementation of the insertion and these time saving procedures are also useful as building blocks for other methods dealing with any arc orienteering problem.

This metaheuristic can also be used to solve arc orienteering problems (AOP). The AOP solutions obtained by the ILS on 50 benchmark instances with an average computation time of less than 1 second are proven to be optimal solutions. We also demonstrated that the performance of the algorithm is not sensitive to small changes in the parameter settings. This indicates that robust behavior might therefore be expected when incorporating this algorithm into real-life applications.

The proposed algorithm is ready to be incorporated in a web-based cycle route planner such as the one presented by Souffriau et al. [22]. The user can specify his home location and route preferences (e.g. the maximum distance between the home location and the starting point, the minimum route distance, the maximum route distance, steep inclines wanted or not, dirt roads wanted or not, rural area or urban area). Afterwards, the planner automatically selects the best starting location and provides the route best tailored to the user's preferences. Finally, the route can be stored to a personal navigation device or the signboard ids can be send as an SMS to the user.

Apart from looking out for new time savings procedures, further research could focus on implementing the proposed algorithm on a larger cycle network, such as the cycle network of Flanders, where the score represents the actual properties of the arc. However, from a practical point of view, cycling more than 100 kilometres on a cycle network seems rather unrealistic given the fact that cyclist are cycling with tourist intentions. Therefore, using a larger network such as the whole region of Flanders with the same upper limit on the distance should not severely hamper our performance since a very large part of this this larger network would not require any exploration by the insert move. However, this increases the practical value of the tool as it allows the planning of cycle trips along and across province borders. On the other hand, using real-world score information might lead to interesting cases as arcs situated in a certain area might have the same or similar scores. It can be interesting to incorporate these aspects into a solution method in order to obtain solutions of a higher quality.

Incorporating the selection of hotels as presented for the (node) orienteering problem by Divsalar et al. [11] into the cycle trip problem might certainly be an interesting research opportunity of high practical value. Finally, the mixed orienteering problem as discussed by Vansteenwegen et al. [27] can be considered. In this extension scores are associated to vertices as well as to arcs. This is useful as some tourist attractions can only be visited at specific vertices and others are visited by traversing particular arcs.

Acknowledgments

This research was partially funded by the Agency for Innovation by Science and Technology in Flanders (IWT)

[1] Aráoz, J., Fernández, E., & Franquesa, C. (2009). The clustered prize-collecting arc-routing problem. *Transportation Science*, *43*, 287–300.

[2] Aráoz, J., Fernández, E., & Meza, O. (2009). Solving the prize-collecting rural postman problem. *European Journal of Operational Research*, *196*, 886–896.

[3] Aráoz, J., Fernández, E., & Zoltan, C. (2006). Privatized rural postman problems. *Computers and Operations Research*, *33*, 3432–3449.

[4] Archetti, C., Feillet, D., Hertz, A., & Speranza, M. G. (2010). The undirected capacitated arc routing problem with profits. *Computers and Operations Research*, *37*, 1860–1869.

[5] Archetti, C., Guastaroba, G., & Speranza, M. G. (2014). An ILP-refined tabu search for the directed profitable rural postman problem. *Discrete Applied Mathematics*, *163(1)*, 3–16.

[6] Archetti, C., Speranza, M. G., Corberán, A., Sanchis, J. M., & Plana, I. (2013). The team orienteering arc routing problem. *Transportation Science*, *doi:10.1287/trsc.2013.0484*, 1–16.

[7] Dantzig, G. B., Fulkerson, D. R., & Johnson, S. M. (1954). Solution of a large-scale traveling salesman problem. *Operations Research*, *2*, 393–410.

[8] Dantzig, G. B., Fulkerson, D. R., & Johnson, S. M. (1959). On a linear-programming, combinatorial approach to the traveling-salesman problem. *Operations Research*, *7*, pp. 58–66.

[9] Deitch, R., & Ladany, S. P. (2000). The one-period bus touring problem: Solved by an effective heuristic for the orienteering tour problem and improvement algorithm. *European Journal of Operational Research*, *127*, 69 – 77.

[10] Dezs, B., Jüttner, A., & Kovács, P. (2011). Lemon - an open source c++ graph template library. *Electron. Notes Theor. Comput. Sci.*, *264*, 23–45.

[11] Divsalar, A., Vansteenwegen, P., & Cattrysse, D. (2013). A variable neighborhood search method for the orienteering problem with hotel selection. *International journal production economics*, *145*, 150–160.

[12] Dror, M. (2000). *Arc Routing. Theory, Solutions and Applications*. Kluwer Academic Publishers, Boston.

[13] Feillet, D., Dejax, P., & Gendreau, M. (2005). The profitable arc tour problem: solution with a branch-and-price algorithm. *Transportation Science*, *39*, 539–552.

[14] Figliozzi, M. A. (2012). The time dependent vehicle routing problem with time windows: Benchmark problems, an efficient solution algorithm, and solution characteristics. *Transportation Research Part E: Logistics and Transportation Review*, *48*, 616 – 636.

[15] Garcia, A., Vansteenwegen, P., Arbelaitz, O., Souffriau, W., & Linaza, M. (2013). Integrating Public Transportation in Personalised Electronic Tourist Guides. *Computers & Operations Research*, *40*, 758–774.

[16] Gendreau, M., Laporte, G., & Semet, F. (1998). A branch-and-cut algorithm for the undirected selective travelling salesman problem. *Networks*, *32*, 263–273.

[17] Ghiani, G., Laganá, D., & Laporte, G. (2007). A branch-and-cut algorithm for the undirected capacitated arc routing problem. *The Journal of the Operations Research Society of America*, (pp. 1–21).

[18] Hao, J., & Orlin, J. B. (1994). A faster algorithm for finding the minimum cut in a directed graph. *JOURNAL OF ALGORITHMS*, *17*, 424–446.

[19] Lourenço, H., Martin, O., & Stútzle, T. (2010). Handbook of metaheuristics 2nd. edition. vol.146. chapter Iterated Local Search: Framework and Applications.. (pp. 363–397). Springer New York, International Series in Operations Research & Management Science.

[20] Miller, C., Tucker, A., & Zemlin, R. (1960). Integer programming formulations and travelling salesman problems. *Journal of the ACM*, *7*, 326–329.

[21] Schilde, M., Doerner, K., Hartl, R., & Kiechle, G. (2009). Metaheuristics for the biobjective orienteering problem. *Swarm Intelligence*, *3*, 179–201.

[22] Souffriau, W., Vansteenwegen, P., Vanden Berghe, G., & Van Oudheusden, D. (2011). The planning of Cycle Trips in the Province of East Flanders. *OMEGA: International Journal of Management Science*, *39*, 209–213.

[23] Souffriau, W., Vansteenwegen, P., Vanden Berghe, G., & Van Oudheusden, D. (2013). The multiconstraint team orienteering problem with multiple time windows. *Transportation Science*, *47 (1)*, 53–63.

[24] Souffriau, W., Vansteenwegen, P., Vertommen, J., Vanden Berghe, G., & Van Oudheusden, D. (2008). A personalised tourist trip design algorithm for mobile tourist guides. *Applied Artificial Intelligence*, *22*, 964–985.

[25] Sörensen, K., Sevaux, M., & Schittekat, P. (2008). Adaptive and multilevel metaheuristics. chapter Multiple neighbourhood search in commercial VRP packages: evolving towards self-adaptive methods. (pp. 239–253). London: Springer volume 136 of *Lecture Notes in Economics and Mathematical Systems*.

[26] Tsiligirides, T. (1984). Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, *35 (9)*, 797–809.

[27] Vansteenwegen, P., Souffriau, W., & Van Oudheusden, D. (2011). The orienteering problem: a survey. *European Journal of Operational Research*, *209*, 1–10.

[28] Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., & Van Oudheusden, D. (2009). Iterated Local Search for the Team Orienteering Problem with Time Windows. *Computers & Operations Research*, *36*, 3281–3290.

[29] Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., & Van Oudheusden, D. (2011). The City Trip Planner: A Tourist Expert System. *Expert Systems with Applications*, *38*, 6540–6546.

[30] Zachariadis, E., & Kiranoudis, C. (2011). Local search for the undirected capacitated arc routing problem with profits. *European Journal of Operational Research*, *210*, 358–367.